VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace vybrané metody vizualizace sítí v prostředí webu

Implementation of a Selected Method of Network Visualization for Web

2020 Lukáš Papík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Papík**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Implementace vybrané metody vizualizace sítí v prostředí webu
Implementation of a Selected Method of Network Visualization for Web

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je implementace vybrané metody vizualizace malých sítí (s přibližně tisíci vrcholy) s podporou uživatelské interakce pro využití ve webových aplikacích.

1. Rešerše obdobných řešení.
2. Implementace metody s použitím vhodně zvolené technologie pro vývoj webových aplikací.
3. Návrh a implementace uživatelského rozhraní pro testování implementovaného řešení.
4. Dokumentace s využitím standardů softwarového inženýrství.

Seznam doporučené odborné literatury:

[1] Barabási, A. L. (2016). Network science. Cambridge university press (https://networksciencebook.com/).

Dále podle pokynů vedoucího práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Miloš Kudělka, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. května 2020 .....

Abstrakt

Cílem této práce je vytvořit webovou aplikaci pro vizualizaci malých sítí s asi přibližně tisíci vrcholy v prostředí webu. Aplikace umožňuje pracovat s vlastní sítí, upravovat vlastnosti vrcholů a hran, implementuje několik metod pro vizualizaci sítě, rozdělení do skupin, detekce komunit, vizualizace plochy skupin nebo export změněné sítě pro použití v jiných aplikacích. Pro práci se sítí je použita knihovna SigmaJs v kombinaci s knihovnou React, pro uživatelské rozhraní.

Klíčová slova: síť; graf; web; vizualizace; bakalářská práce

Abstract

The aim of this work is to create a web application for visualization of small networks, with about a thousand vertices, in a web environment. The application allows us to work with our own network, edit the properties of vertices and edges, implements several methods for network visualisation, splitting into groups, community detection, visualizing a group area or exporting a modified network for use in other applications. For the work with networks is used the SigmaJs library in combination with the React, for the user interface.

Keywords: network; graph; web; visualization; bachelor's thesis

Obsah

Seznam použitých zkratek a symbolů

BLOB	–	Rozsáhlý binární objekt
CSS	–	Kaskádové styly
DOM	–	Objektový model dokumentu
ECMAScript	–	Specifikace skriptovacího jazyka standardizovaného organizací ECMA
GEXF	–	XML formát určený pro zápis grafu
GML	–	Formát pro zápis grafu
HTML	–	Hypertextový značkovací jazyk
JSON	–	Objektová notace v Javascriptu
JSX	–	Syntaktické rozšíření Javascriptu o XML elementy
SVG	–	Škálovatelná vektorová grafika
UML	–	Unifikovaný modelovací jazyk
URL	–	Jednotná adresa zdroje
W3C	–	World Wide Web konsorcium
XML	–	Rozšiřitelný značkovací jazyk

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

S každým dnem přibývá množství vytvořených dat. Data, informace jsou shromažďovány již v téměř každém odvětví, ať už je to zdravotnictví, ekonomika nebo sociální vědy. Každý se snaží shromáždit co nejvíce informací, které by bylo možné dále zpracovat a následně vyhodnocovat. Když už ale máme dostatek informací, v sumarizované, organizované formě, nastává otázka, jak tyto informace reprezentovat.

Existuje několik možností. Textová podoba, se kterou se nejčastěji setkáme jako věty či odstavce. Tato forma se používá pokud chceme zdůraznit určitou informaci a zároveň text obsahuje málo hodnot na použití tabulky nebo grafu. Tabulky, kde informace jsou reprezentovány v sloupcích a řádcích umožňují zobrazit různé jednotky nebo hodnoty, které jsou obtížněji vizualizovatelné a je snazší porovnávat kvantitativní hodnoty. Konečně graficky, pomocí grafů, kterých existuje celá řada, například rozptylový, časový, sloupcový nebo síťový. Lze v nich spatřit trendy, vzory, které mohou mít vliv na celý systém nebo jeho chování. Často jsou tyto trendy či vzory obtížně spatřitelné z předchozích metod vizualizace.

Cílem práce je tedy vytvořit webovou aplikaci, ve které si bude moci uživatel vizualizovat síť a na tuto síť aplikovat vybranou metodu vizualizace. Druhá kapitola se zaobírá základy z oblasti teorie grafů a sítí, které se mohou vyskytovat v této práci, třetí kapitola potom popisuje již existující řešení z oblasti sítí, jak už hotové aplikace, tak knihovny, které lze použít. Ve čtvrté kapitole jsou popsány požadované aspekty aplikace, technická stránka webové aplikace, diagram případů užití s jedním scénářem, použité technologie, knihovny, problémy během implementace, diagram komponent zobrazující celý systém a systémové požadavky. V poslední, páté kapitole je popsáno všeobecně uživatelské rozhraní aplikace. Příloha A obsahuje postup kroků pro nasazení.

2 Teorie

Popis základních pojmů z oblasti teorie grafů a sítí, které se vyskytují v následujících kapitolách této práce.

Graf je definovaný jako dvojice množin V a E, kde V reprezentuje vrcholy a E hrany grafu.

Hrana je uspořádaná, nebo neuspořádaná dvojice vrcholů.

Síť ve vědecké literatuře jsou pojmy síť a graf zaměnitelné. [1]

Orientovaný graf označujeme graf takový, ve kterém jsou hrany uspořádané dvojice, kde jeden vrchol označíme za počáteční a druhý za koncový. Jinak je graf neorientovaný.

Acyklický graf označuje graf takový, který neobsahuje cyklus, kružnici.

Úplný graf je graf, ve kterém jsou všechny dvojice vrcholů spojeny hranou.

Vážené sítě jsou sítě, ve kterých je každá hrana ohodnocena číselnou hodnotou[1].

Komponenta je taková podmnožina vrcholů, ve které pro každý vrchol existuje cesta ke všem ostatním vrcholům z této podmnožiny a zároveň tato podmnožina není součástí jiné, větší množiny s vlastností, že každý vrchol může mít cestu ke všem ostatním vrcholům. [2]

Shluk neboli komunita nebo modul, je skupina vrcholů grafu, které pravděpodobně sdílí podobné vlastnosti nebo hrají v grafu podobnou roli. [3]

Stupeň vrcholu udává počet vrcholů, se kterými má vrchol společnou hranu.

Sousední vrcholy označujeme vrcholy, které jsou s daným vrcholem spojené hranou.

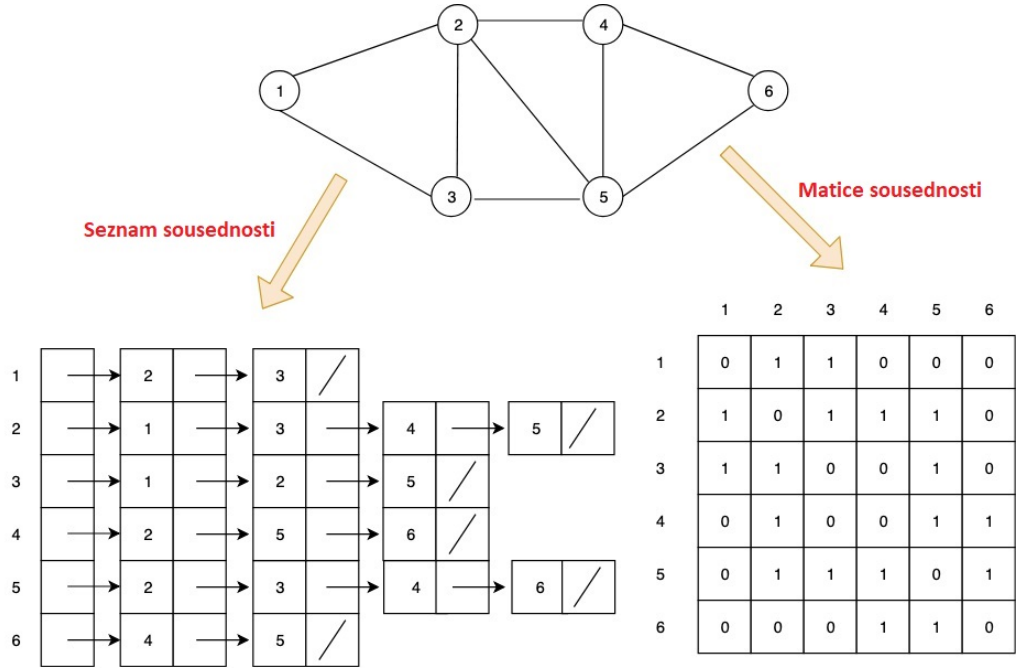
List je vrchol, který má stupeň vrcholu 1.

2.1 Reprezentace sítě

Obecný popis grafu, tedy množiny vrcholů, hran a incidenční zobrazení. Nejčastěji se setkáme s popisem grafu pomocí matic nebo pomocí seznamů, které jsou vhodnější pro počítačové zpracování. [4]

2.1.1 Matice sousednosti

Matice sousednosti je čtvercová matice, kde řádky i sloupce reprezentují vrcholy sítě (viz obr. 1). Pro nevážené sítě hodnota 1 na dané pozici v matici bude znamenat, že příslušné vrcholy spolu sousedí, jinak na této pozici bude hodnota 0. U vážených sítí je místo hodnoty 1 použitá konkrétní váha hrany. Pro neorientovaný graf bude matice symetrická a vystačili bychom si jen s trojúhelníkovou maticí.[4] Paměťová složitost je $O(|V|^2)$, kontrola jestli je mezi vrcholy hrana



Obrázek 1: Reprezentace neorientovaného grafu pomocí matice sousednosti a seznamu sousedností [5]

či odebírání, nebo přidávání hrany má složitost pouze $O(1)$. Vhodné pro úplné grafy nebo grafy s velkým počtem hran.

2.1.2 Matice incidence

Je reprezentace pomocí matice o velikosti $m \times n$, kde m reprezentuje vrcholy, odpovídá počtu vrcholů, n reprezentuje hrany, odpovídá počtu hran. U orientovaných grafů udává incidenci v matici hodnota 1 pro počáteční vrchol a konečný vrchol hodnota -1, hodnota 2 označuje smyčku a hodnota 0 pro zbytek případů. U neorientovaných sítí se používají pouze dvě hodnoty 0 a 1. [4] Tento způsob reprezentace sítě zde zmiňuji pouze pro úplnost, jinak jí v práci nepoužívám.

2.1.3 Seznam sousednosti

Pro každý vrchol v síti existuje seznam obsahující vrcholy s ním sousedící (viz obr. 1). U vážené sítě bychom museli pro každého souseda v seznamu evidovat i váhu hrany. Oproti matici sousednosti, kde uchováváme informace i o hranách vrcholů, které spolu nesousedí, tak v této reprezentaci uchováváme v paměti pouze hrany se kterými vrchol sousedí.[4] Tato reprezentace je pro nás výhodnější, pokud má graf menší počet hran. Paměťová složitost je $O(|V| + |E|)$.

2.2 Metoda vizualizace

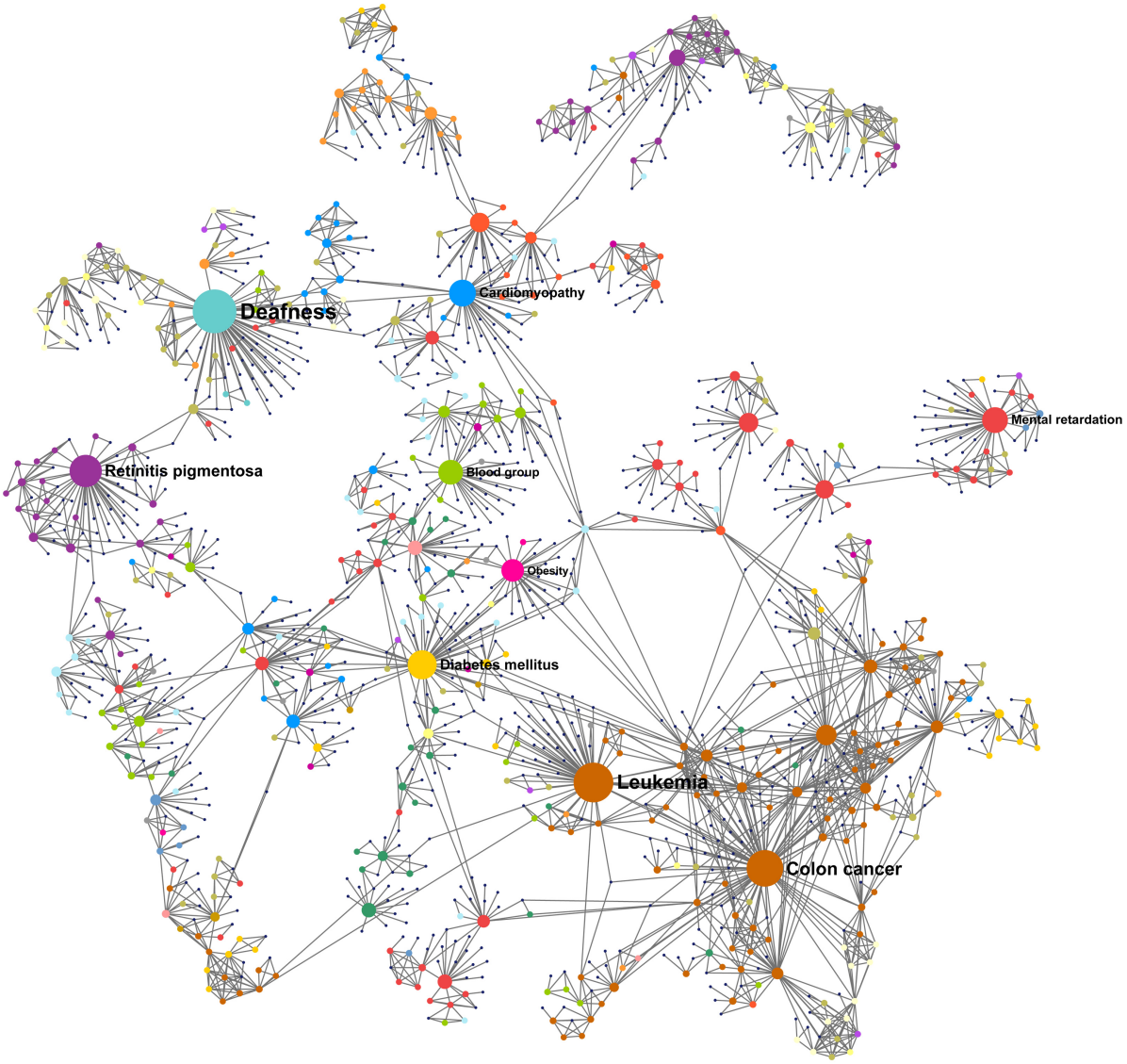
Metoda, která poskytuje na základě typu zobrazení vizuálně přitažlivý způsob prezentace struktury sítě. Každé zobrazení zdůrazňuje nějakou vlastnost sítě vhodnou pro konkrétní aplikaci, oblast nebo typ dat. U metod často záleží i na estetice a použitelnosti, proto se jejich kvalita určuje podle mnohých kritérií. Pro určení kvality tedy často záleží, jak moc se kříží hrany, vzdálenost mezi vrcholy, symetričnost, tvar hran, úhel hran, jestli má vrchol mnoho hran, kompaktnost, tedy velikost zabraného místa grafem apod. [6] Následující metody patří mezi nejvíce používané.

2.2.1 Force-based zobrazení

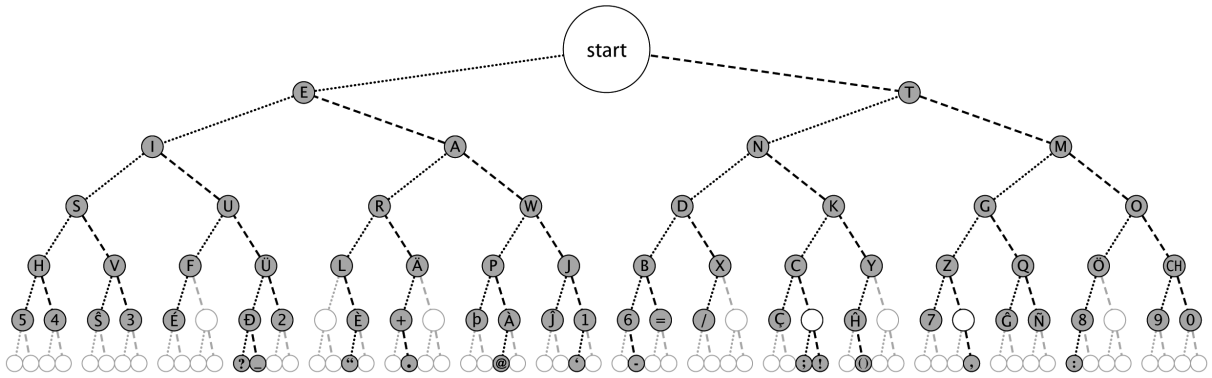
Skupina algoritmů, které patří k nejvíce flexibilním metodám pro výpočty zobrazení menších neorientovaných sítí. Flexibilita je dána hlavně tím, že pro výpočty se používají pouze informace ze struktury grafu. Často se přirovnávají k fyzikálním simulacím, např. když si představíme vrcholy jako elektricky nabité částice, kterých odpudivá síla působí na všechny ostatní vrcholy a přitažlivá síla působí mezi vrcholy, které spolu sousedí. Tyto síly působí na vrcholy a vytvářejí tak pohyb. Ve všeobecnosti tyto algoritmy definují funkci, která mapuje každé rozložení jako číslo reprezentující energii rozložení. Tato funkce je definována tak, že nižší energie odpovídá rozložení, ve kterém jsou sousední vrcholy blízko určité předem stanovené vzdálenosti jeden od druhého a vzdálenější vrcholy jsou odděleny. Výsledné rozložení grafu potom odpovídá minimu této funkce. Jsou příjemné na pohled, vykazují symetrii a snaží se o nejmenší počet křížení hran. Jejich nevýhodou je dlouhý čas uspořádání vrcholů, protože se musí v každé iteraci vypočítat pro každý vrchol jeho síla v závislosti na celé síti. [7] Tento problém řeší Barnes-Hut simulace, se kterou dosahují algoritmy složitost $O(nlog(n))$, což je patrné zejména u větších sítí. [8] Mezi nejznámější *silově založené* algoritmy patří ForceAtlas, ForceAtlas2, Yifan Hu, Fruchterman-Reingold nebo OpenOrd. Jsou vhodné pro vizualizaci např. biomedicínských dat (viz obr. 2) nebo sociální sítě, ve kterých je větší množství listů. [6]

2.2.2 Ortogonální zobrazení

V tomto zobrazení se snažíme uspořádat vrcholy tak, aby všechny hrany měly úhel pouze násobky úhlu $\frac{\pi}{2}$ a byly tedy vodorovné nebo svislé. Pokud bychom chtěli, aby nedocházelo ke křížení hran, bylo by nutné aby vrcholy byly maximálně stupně čtyři. Minimalizováním počtu křížících se hran a velikosti zabraného místa můžeme zvýšit čitelnost výsledné sítě, o co se snaží velké množství algoritmů, jako například HOLA: Human-like Orthogonal Network Layout [10] nebo Graph Compact Orthogonal Layout [11]. U jednotlivých algoritmů ale záleží na konkretním výsledku, protože mnohdy je nutná ruční úprava pozic vrcholů pro lepší výsledek. Tato metoda je populární u diagramů pro softwarové inženýrství, jako jsou například třídní diagramy, ER diagramy apod. [6]



Obrázek 2: Síť zobrazující geny lidských poruch spojené se známými onemocněními označující společný genetický původ těchto nemocí s použitím algoritmu Force-Atlas2 [9]



Obrázek 3: Binární strom Morseovy abecedy [12]

2.2.3 Stromové zobrazení

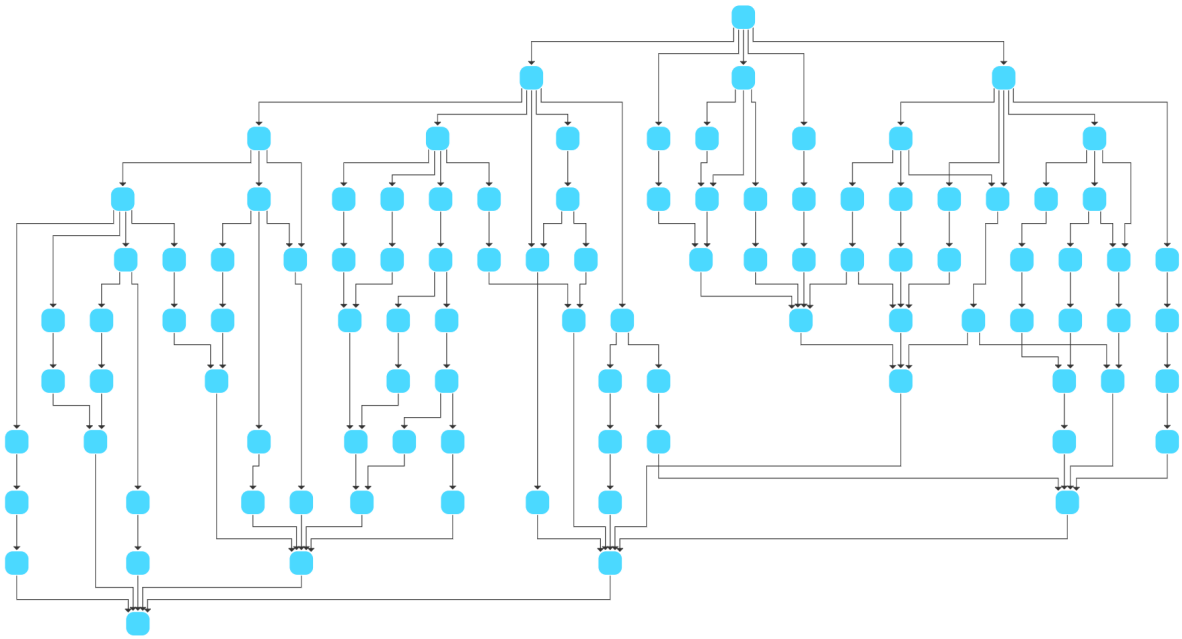
Zobrazení souvislých acyklických grafů s typickou hierarchickou strukturou dat (viz obr. 3). Při vizualizaci nám záleží na vícero vlastnostech, jako např. umístění potomků, ohýbaní hran, křížení hran, chceme aby vrcholy stejného stupně byly zarovnané na stejné úrovni, udržovaly relativní uspořádaní pro levé a pravé podstromy, symetričnost, vzdálenost nebo počet listů. Pro vizualizaci existuje mnoho přístupů, v závislosti na požadovaném typu stromu nebo estetickém výsledku. Často se používá pro reprezentaci dědičnosti tříd v objektově orientovaném programování, souborového systému, struktury webu, evolučních sítí, výsledky turnajů nebo rozhodovacích stromů. [6]

2.2.4 Vrstvené zobrazení

Slouží pro vizualizaci orientovaných síti (viz obr. 4). Síť se rozdělí na vrstvy tvořené z množin vrcholů a v jednotlivých vrstvách je propojení mezi vrstvami a uspořádaní vrcholů v jednotlivých vrstvách dáno konkrétní metodou. Sugiyamaova metoda [13], nazvaná podle autora tohoto zobrazení, se skládá ze čtyř kroků, kdy se nejprve odstraní cykly ze sítě, přidělí jednotlivé vrcholy sítě do vrstev, vrcholy se uspořádají ve vrstvách a konečně se každému vrcholu sítě přiřadí jeho pozice, pro co nejlepší výsledek. Z estetického hlediska se snaží o to, aby hrany směřovali jedním směrem a byly pokud možno rovné, délka hran byla co nejmenší, pro co nejvyšší citelnost, rovnoměrně rozložit vrcholy a snížit počet křížení hran, pro zabránění nejasností. Toto zobrazení je vhodné pro diagramy závislosti, diagramy průběhu nebo návrhy některých polovodičových integrovaných obvodů. [6]

2.2.5 Obloukové zobrazení

V tomto zobrazení jsou vrcholy umístěny na horizontální, nebo vertikální ose a hrany jsou reprezentovány jako elipsy (viz obr. 5). Hrany mohou být silnější u vážených sítí, v závislosti na jejich váze. Pro lepší přehled a zvýraznění jednotlivých shluků je důležitá pozice jednotlivých

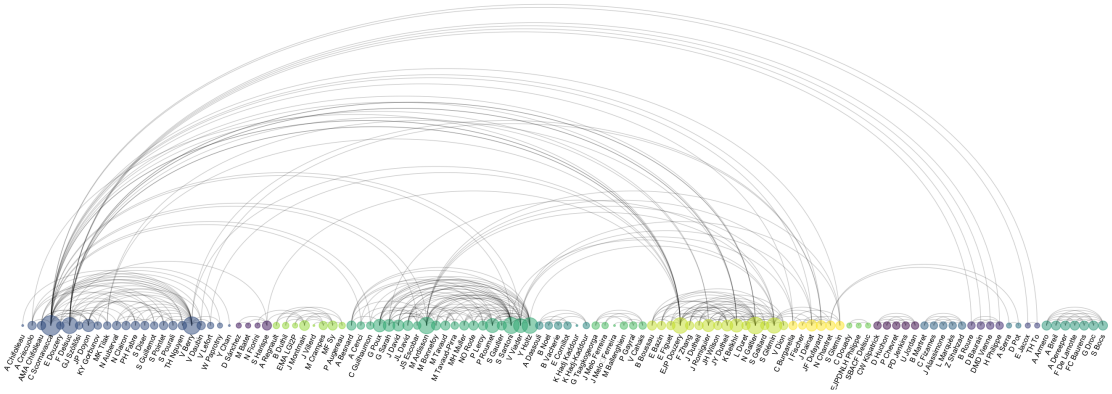


Obrázek 4: Jednoduché vrstvené rozložení [14]

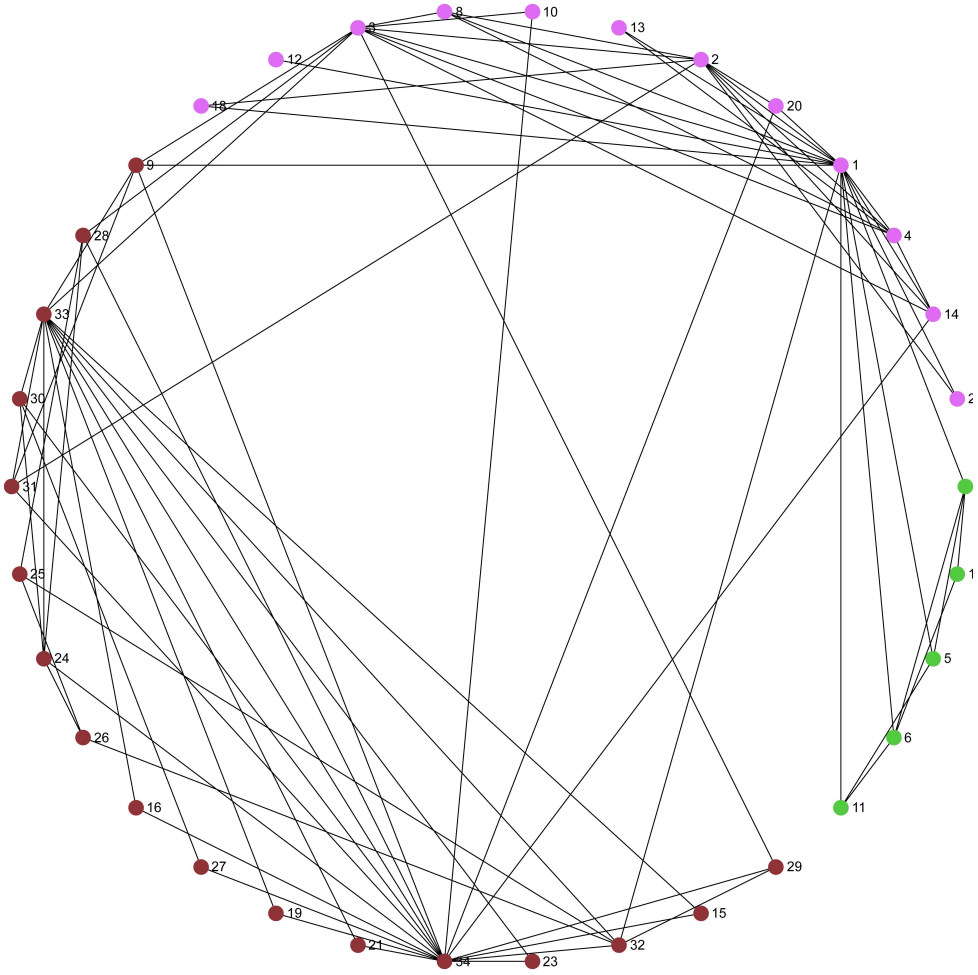
vrcholů. Výhodou je možnost zobrazit popisky všech vrcholů, co není úplně možné u všech typů zobrazení. [15]

2.2.6 Kruhové zobrazení

Zobrazení, ve kterém jsou jednotlivé vrcholy umístěné rovnoměrně na kružnici, takže tvoří body pravidelného konvexního mnohoúhelníku a hrany jsou rovné (viz obr. 6). Kružnic v síti může být vícero, závisí na použitém algoritmu. Algoritmy pro toto zobrazení si dávají za cíl minimalizovat počet křížících se hran. Struktura sítě může být zvýrazněna tím, že detekujeme komunity v síti a jednotlivé vrcholy z těchto komunit umístíme postupně za sebou, čímž rovněž můžeme snížit počet křížení hran. Jsou vhodné pro vizualizaci síťových topologií, sociálních skupin nebo biologických sítí. [6]



Obrázek 5: Obloukové zobrazení spoluprací vědeckých pracovníků [15]



Obrázek 6: Zachary karate klub s detekovanými komunitami

3 Existující řešení

Existuje mnoho nástrojů pro práci se sítěmi, avšak tyto nástroje jsou často navržené jako tzv. desktopové aplikace. Hlavně kvůli tomu, že mohou využit veškerý výpočetní výkonu počítače, zejména procesoru počítače. Avšak s rozvojem webu a nástupem HTML5, které přináší spoustu nových možností, je možné například využít více vláken i v rámci webového prohlížeče, resp. Javascriptu, pomocí web workerů (kapitola 4.1.2). Použití webových aplikací nese rovněž i další výhody. Aplikace může fungovat bez nutnosti instalace dalších programů či doplňků do prohlížeče, fungovat bez připojení k internetu nebo vypadat jako desktopová aplikace. Taktéž nemusí běžet jen na počítači, ale i na jiných zařízeních, které jsou dnes dostatečně výkonné na to, aby zvládli provádět náročnější výpočty. V našem případě budou nejnáročnější výpočty u metod vizualizace nebo při určení komunit v síti. Níže uvedené aplikace a knihovny slouží pro práci se sítěmi, snažil jsem se vybrat ty nejpopulárnější a stručně je charakterizovat.

3.1 Desktopové aplikace

Aplikace, které vyžadují stažení spustitelného souboru nebo instalátoru na lokálním počítači.

3.1.1 Gephi

Mezi nejpopulárnější multiplatformní program pro práci se sítí a její vizualizací patří Gephi (viz obr. 7), který dokáže pracovat s většími sítěmi, specifikace programu udává desítky až stovky tisíc vrcholů a až miliardu hran. Poskytuje mnoho různých funkcí, aplikace různých metod vizualizace na síť, převažně typu Force-Based (viz kapitola 2.2.1), výpočet blízkosti, centrality, shlukovacího koeficientu, detekci komunit, hledání nejkratší cesty, vývoj sítě v čase, obarvování nebo dynamické filtry. Program také podporuje rozšíření v podobě pluginů a pracuje kromě jiných i s vlastním formátem pro popis sítě GEXF (viz kapitola 4.2.3), který vychází z XML formátu. Nevýhodou je nutnost běhového prostředí Java. [16]

3.1.2 Pajek

Starší program pro analýzu a vizualizaci velkých sítí, které mohou obsahovat až miliardu vrcholů, limitem je často pouze velikost operační paměti počítače. Často se nazývá síťovou kalkulačkou, protože pro získání výsledků je nutná posloupnost kroků a operací, ty z této aplikace dělají silný nástroj. Dává si za cíl rozdělit velké sítě na menší, na které lze aplikovat více sofistikovanější metody. Dále implementuje algoritmy pro vizualizaci, jako například Kamada-Kawai, Fruchterman-Reingold (viz kapitola 4.2.9) nebo FishEye zobrazení. Nevýhodou je starší, neintuitivní uživatelské rozhraní a funkčnost pouze na operačním systému Windows. [17]

3.1.3 Cytoscape

Přestože byl Cytoscape (viz obr. 8) původně navržen pro biologický výzkum, profilování genového výrazu, interakce mezi buňkami a jiné, nyní je obecnou platformou pro komplexní síťovou analýzu a vizualizaci. Ihned po instalaci je dostupné kruhové (viz kapitola 2.2.6), Force-Based (viz kapitola 2.2.1) nebo vrstvené zobrazení (viz kapitola 2.2.4). Základní verzi lze rozšířit o rozšíření, které přidávají další funkcionalitu, kupříkladu ortogonální (viz kapitola 2.2.2) nebo stromové rozložení (viz kapitola 2.2.3). Nevýhodou je nutnost platformy Java a stále existuje mnoho funkcí, rozšíření, které se zaměřují na doménu bioinformatiky. [18]

3.2 Webové aplikace

Aplikace, které pro svojí funkčnost potřebují pouze webový prohlížeč. U těchto aplikací jsem se setkal mnohokrát s tím, že podporovaly výhradně formáty z tabulkových procesorů, které si myslím, že u běžných uživatelů budou jedny z nejvíce používaných pro práci s daty.

3.2.1 RAWGraphs

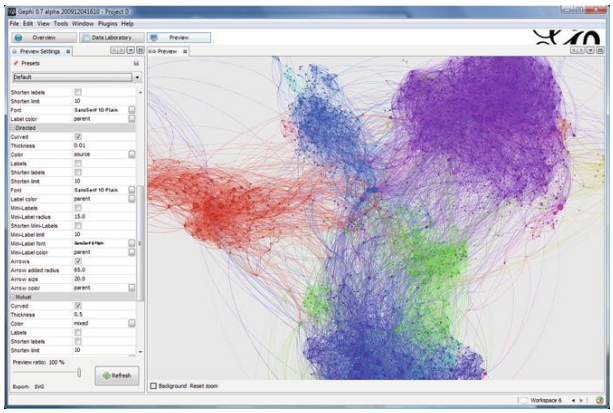
Webová aplikace (viz obr. 9) s otevřeným zdrojovým kódem pro vizualizaci dat, která se snaží udělat vizualizaci co nejpřívětivější. Stačí vybrat soubor s daty, vhodnou vizualizaci, případně je možné vytvořit svou, a poté stačí přetáhnou myší názvy sloupců dat, podle jejich datového typu, na jednotlivé osy, v závislosti na typu vizualizace. Jednotlivé vizualizaci můžeme dále upravovat podle možností které poskytují. Nevýhodou je menší paleta dostupných vizualizací a jejich nastavení. [19]

3.2.2 Rhumbl

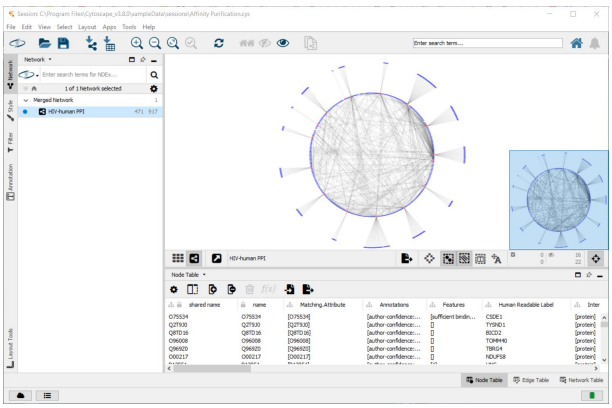
Webová aplikace (viz obr. 10) pro vizualizaci dat, která poskytuje oproti všem zmíněným nejméně funkcí. Ve verzi zdarma poskytuje možnost vizualizovat jednu síť, která se uloží na server. Může pracovat se sítěmi, které jsou reprezentovány buď jako seznam sousedností nebo matice sousednosti a veškerý výpočet zobrazení probíhá na serveru. Obsahuje vyhledávání vrcholů, kdy po zadání názvu se zobrazí náhled vyhledávaného vrcholu a jeho sousedů. Nutno podotknou že tato aplikace je stále ve vývoji. Hlavní nevýhodou je potřeba nahrávat data na server, ve volné verzi uložení pouze jedné sítě a nedostatek funkcí. [20]

3.3 Knihovny

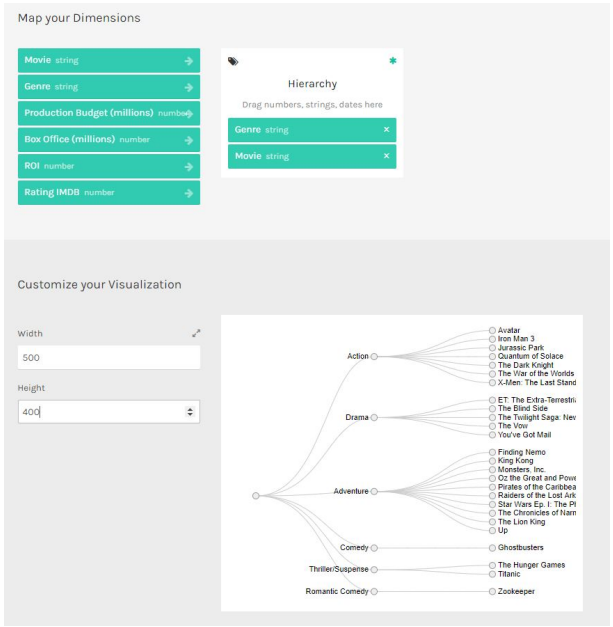
Poskytují hotové řešení, nástroje na určité problémy během vývoje. Přestože existuje knihoven na tuto problematiku mnoho, tak jsem se snažil vybrat několik nejpopulárnějších. Vzhledem k vývoji webové aplikace jsou následující knihovny napsány výhradně v jazyku Javascript (kapitola 4.1.1) a mají otevřený zdrojový kód.



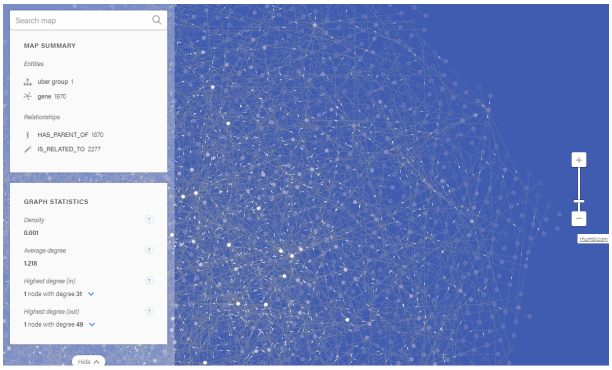
Obrázek 7: Ukázka programu Gephi



Obrázek 8: Ukázka programu Cytoscape



Obrázek 9: Ukázka programu RAWGraphs



Obrázek 10: Ukázka programu Rhumbl

3.3.1 D3.js

Pravděpodobně nejznámější a nejvíce využívaná knihovna pro dynamické a interaktivní vizualizace dat na webu. Využívá pouze webové stadardy, kombinuje technologie HTML, CSS a SVG, se kterými manipuluje pomocí rozhraní DOM. Jednoduchou tabulku je možné vytvořit například z pouhého pole čísel, nebo použít stejné pole pro interaktivní vizualizaci. Poskytuje široké množství funkcí, podporuje animace, interakci, zobrazení a zpracování většího množství dat. Všechny tyto a mnoho dalších funkcí je možných zejména kvůli tomu, že je tato knihovna rozdělena na moduly, které lze kombinovat a znovu používat. Nevýhodou je přílišná komplexnost a tedy nutná znalost velkého množství funkcí. [21]

3.3.2 Cytoscape.js

Moderní knihovna, která vznikla jako náhrada původní verze Cytoscape web [22], z vývojového hlediska je jedna z nejaktivnějších, čemuž nasvědčuje i množstvo funkcí, které v základu poskytuje. Zaměřuje se nejen na vizualizaci, ale i na analýzu menších sítí. Podporuje běh jak na straně klienta, tak i serveru, může fungovat i bez prohlížeče, animace, interakci se sítí, podpora jazyka R, pracuje primárně s formátem JSON, skvělá dokumentace a disponuje i možností rozšířit funkcionalitu vlastními nebo existujicími rozšířeními. [23]

3.3.3 sigmaJs

Knihovna pro vizualizaci sítí na webu, která v základu poskytuje vykreslování sítě na HTML element plátna nebo pomocí WebGL a podporu vstupů, reakce na události vstupu a sítě, např. přejíždění myší nad vrcholem, hranou, oddalování grafu či jiné. Hlavní předností je, že podporuje rozšíření, které mají otevřený zdrojový kód, takže je možné dotvářet vlastní funkcionalitu sítě, avšak některé rozšíření nefungují s vykreslováním pomocí WebGL. V minulosti vznikaly i odvozené knihovny, založené na této knihovně, v době psaní této práce jsem žádnou aktivně vyvíjenou nenašel. Vcelku dobrá dokumentace, jednoduché použití, možnost vlastních úprav, jak Sigmy samotné, tak i rozšíření a široká komunita byly důvody, proč jsem se rozhodl použít tuto knihovnu i přes to, že tato knihovna nebyla skoro dva roky aktualizována.

4 Aplikace

Cílem práce bylo tedy vytvořit webovou aplikaci pro vizualizaci sítí, ve které si bude moct uživatel načíst vlastní sítě z již existujících aplikací pro zpracování sítí a poté na tyto načtené sítě aplikovat vybranou metodu vizualizace. Následně mít možnost upravovat nastavení vybrané vizualizace, tak i vrcholů a hran, pomocí uživatelského rozhraní. Mimo to měnit umístění vrcholů v síti, rozdělit síť na skupiny, tj. množinu vrcholů, na kterou můžeme následně hromadně aplikovat nastavení, zobrazit plochu, kterou zabírá určitá skupina a detekovat skupiny v síti. Aplikace také dále umožňuje exportovat výslednou síť zpátky do formátů, které podporují jiné aplikace nebo exportovat pouze grafickou část sítě, jako obrázek sítě.

4.1 Použité technologie

Vzhledem k tomu, že tato aplikace je webová, bylo nutné zvolit patřičné technologie a nástroje pro vývoj. Jak již bylo zmíněno, existuje mnoho knihoven pro práci se sítěmi na webu, mnohdy jsou až příliš komplexní, nemají dostatečnou dokumentaci nebo postrádají aktivní komunitu, která by vám poradila v případě problému. Pro manipulaci se sítěmi mi přišla nejvhodnější zmíněná knihovna SigmaJs (viz kapitola 3.3.3). Další technologie, knihovny které jsem použil jsou popsány v této kapitole. Všechen použitý software je s otevřeným zdrojovým kódem a svobodnou softwarovou licencí. Proto jsem použil některé z již existujících rozšíření pro SigmaJs nebo její odvozenin a upravil je pro svou potřebu nebo aby vůbec byly funkční.

4.1.1 Javascript

Javascript je interpretovaný, objektově orientovaný jazyk, který se obvykle spouští na straně klienta v prohlížeči a dokáže manipulovat s webovou stránkou i s prohlížečem. Jeho nevýhodou je, kromě jiných, že pro vykreslování se používá jediné, hlavní vlákno, které během častých a náročných operací, jako je například výpočet pozic v nějaké metodě vizualizace, může být blokováno výpočtem, což vede k tomu, že stránka přestane reagovat a výsledek metody vidíme až po ukončení náročné činnosti. Řešením může být použiti pracovníků (web worker, kapitola 4.1.2), ti jsou dostupní od HTML5.

4.1.2 Web worker

Web worker, pracovník, umožňuje vykonávání skriptů v nových vláknech na pozadí oddělených od hlavního spouštěcího vlákna. Skripty, které má vykonat pracovník musí být jako samostatné soubory. Oddělení od hlavního vlákna, které se většinou stará o překreslení obsahu webové stránky, umožňuje provádět náročnější operace nebo výpočty bez toho, aby došlo k blokaci nebo zpomalovaní uživatelského rozhraní. Nová vlákna jsou tvořena na úrovni operačního systému, nastává tedy možnost, kdy by mohlo dojít k problémům se souběhem. Existují proto jistá omezení pro vlákna na pozadí, ty nemohou přistupovat k objektům DOM, měnit obsah stránky a

přistupovat k některým metodám a vlastnostem globální proměnné *window*, která obsahuje i objekt SigmaJs. Komunikace mezi hlavním vláknem a pracovníkem probíhá pomocí zpráv. [24]

4.1.3 ReactJs

Javascriptova knihovna pro vytváření uživatelským rozhraní. Umožňuje rozdělit uživatelské rozhraní na znovupoužitelné zapouzdřené části, komponenty, kde každá komponenta má svoje vlastnosti a stavy. Hlavní výhodou je, že k překreslení dochází pouze u komponenty, kde se změnil stav nebo vlastnost a proto se nemusí vykreslovat celá stránka znovu. Využívá JSX pro definici uživatelského rozhraní, které se používá hlavně v Reactu.[25]

4.1.4 iwanthue

Knihovna pro generování libovolně velké palety barev, která generuje barvy tak, aby nepůsobily rušivě a zvýšily uživatelský zážitek. Algoritmus se snaží vybírat optimálně odlišné barvy z barevného spektra pro nejlepší čitelnost. Umožňuje nastavit odstín, sytost, jas a počet generovaných barev. [26] Tuto knihovnu používám u detekce komunit, kde se vytvoří paleta barev, podle toho, kolik je komunit a následně se aplikuje tato barva i na vrcholy ve skupinách.

4.1.5 jLouvain

Knihovna, která obsahuje implementaci Louvain algoritmu pro detekci komunit v síti. Napsaná v jazyku Javascript. Funguje na algoritmu popsaném v kapitole 4.2.8. Jako vstup požaduje vrcholy a hrany sítě. Dokáže pracovat i s váženými sítěmi. [27]

4.1.6 react-color

Sada React komponent, které slouží pro výběr konkrétní barvy. Obsahuje mnoho druhů komponent, v mém případě jsem použil dvě komponenty. První zobrazuje odstín, průhlednost a výběr z několika výchozích barev a je použita pro nastavení barvy vrcholů nebo hran. Ve druhé komponentě je pouze výběř odstínu, používá se u nastavení skupin nebo pro barvu konkrétního vrcholu.

4.1.7 PegJs

Knihovna pro generování jednoduchého syntaktického analyzátoru pro jazyk Javascript. Je založená na typu formální gramatiky, kde pro popis struktury souboru je použita množina pravidel. Integruje lexikální i syntaktickou analýzu. Vytvářet i používat pravidla lze ve webovém prohlížeči. [28] Tato knihovna je využívána pro zpracování souborů typu GML.

4.2 Popis implementace aplikace

Na začátku vznikala otázka, zda nebude lepší využít architekturu klient-server, ve které by klient zaslal požadavek, server by ho zpracoval a klientovi by se vrátila odpověď s již vypočtenými hodnotami a sloužil by pouze pro zobrazení výsledků nebo zadávaní požadavků. Ačkoliv React podporuje vykreslování na straně serveru tak, že po zaslání požadavku se ze serveru vrátí hotová stránka, rozhodl jsem se provádět veškerou logiku na straně klienta, tedy webového prohlížeče, jelikož pracujeme s malými sítěmi. Řešení klient-server by bylo vhodné u sítí, které mají vetší rozsah a potřebovali bychom provádět časté a náročné operace.

4.2.1 Single Page App

Tyto *jednostránkové* aplikace fungují tak, že po načtení hlavního HTML souboru se stáhne vše potřebné na to, aby při interakci s aplikací nedocházelo k načítaní dalších HTML souborů. Během manipulace se pouze překreslí obsah načtené stránky, např. při přechodu z úvodní komponenty na hlavní komponentu. Takto funguje i naše aplikace, z tohoto důvodu aplikace obsahuje tlačítko v menu pro návrat zpět na úvodní komponentu. Výhodou je rychlost a odezva aplikace. Na druhou stranu nevýhodou je indexace u vyhledávačů, protože načítaní takovýchto aplikaci je pomalejší než klasický přístup, vzhledem k tomu, že se musí stahovat soubory s vyšší velikostí a proto pozice takovéto aplikace je ve vyhledávači nižší, v našem případě nepodstatné. Ovšem podstatnější nevýhodou je, že při měnění komponent se nemění URL adresa, proto tlačítko zpět a vpřed v prohlížecí ztrácejí podstatu. Hlavně u změny, kdy se překreslí celá obrazovka, takže uživatel má tendenci intuitivně se vrátit s použitím tlačítka zpět. S příchodem HTML5 lze vyřešit použitím History API.

4.2.2 Způsoby importu sítě

Aplikační logiku tvoří Javascript, resp. JSX spolu s CSS pro stylování. Aplikace se skládá z několika částí, resp. komponent. Po otevření stránky se zobrazí jednoduché úvodní menu aplikace. V tomto menu si může uživatel načíst vlastní síť v některém z podporovaných formátů (viz kapitola 4.2.3) nebo vybrat některou z ukázek.

Způsobů pro načítaní sítí je vícero. První možností je, načíst soubor ve zmíněném úvodním menu výběrem lokálního souboru. Pro výběr vlastního souborů má Javascript kvůli bezpečnosti omezený přístup k lokálnímu úložišti. Pro tento účel existuje nový atribut *file* u elementu *input* od verze HTML5, který vytvoří nativní dialog pro výběr souboru a měla by být používaná výhradně tato metoda. Prohlížeč může blokovat nucení uživatele o zvolení souborů v případě bombardování tohoto dialogu nebo blokovat výběr citlivých systémových souborů. Pro práci se soubory po výběru existuje rozhraní *File API*, které ale je v době psaní této práce jen návrh konsorcia W3C a mělo by projít změnou, avšak i přesto je implementované v prohlížečích. [29] Další možnost kterou úvodní menu poskytuje je výběr některé z ukázkových sítí, tyto sítě se nenačítají z lokálního úložiště, ale z webu, proto u nich nenastávají výše zmíněné problémy s

načítáním. Tyto ukázkové sítě jsem našel v repozitáři [9] projektu Gephi. Poslední možností je načíst síť vytvořením URL adresy s parametry. URL může obsahovat pět parametrů, z toho dva jsou povinné a to pro URL sítě a pro její formát. Další tři nepovinné parametry udávají název souboru, možnost skrýt panely s nastavením po načtení a možnost načíst pouze největší, maximální komponentu. Takto načtenou síť je možno vkládat i do jiných webových stránek a pracovat s ní. Následující výpis s číslem 1 zobrazuje použití všech parametrů, parametr n udává URL k síti, t formát načítané sítě, f název souboru, h pokud obsahuje hodnotu 1, tak skryje panely po načtení a m pokud obsahuje hodnotu 1, tak zobrazí pouze největší komponentu. V případě že chybí povinné parametry nebo nastane chyba je zobrazeno úvodní menu a lze použít některou z předchozích metod.

```
http://127.0.0.1:3000/?n=127.0.0.1:3001/sit&t=json&f=nazevSouboru&h=1&m=1
```

Výpis 1: Ukázka importu sítě pomocí URL parametrů

4.2.3 Podporované formáty

Json JavaScript Object Notation je formát pro výměnu dat, který je jednoduše čitelný i zapisovatelný a má podporu pro zpracováni v Javascriptu. Zapisuje se jako kolekce párů název a hodnota. Hodnota může být text, číslo, objekt (množina párů název a hodnota), pole, pravdivostní hodnoty a hodnota null. Tento formát lze importovat i exportovat.

V našem případě je nutné, aby soubor obsahoval seznam vrcholů s patřičnými hodnotami, jako je id, pozice, velikost apod. Dále seznam hran, který musí obsahovat id hrany, počáteční vrchol a konečný vrchol. Nepovinně potom i objekt s nastavením, který je vytvořen při exportu na tento formát. Ukázka i s nastavením ve výpisu č. 2.

```
{
    "nodes": [
        {
            "label": "11",
            "x": -225.27,
            "y": 23.20,
            "id": "11",
            "color": "rgb(0,0,0)",
            "size": 10.0
        }
    ],
    "edges": [
        {
            "source": "11",
```

```
            "target": "11",
            "id": "19",
            "color": "rgb(0,0,0)",
            "size": 1.0
        }
    ],
    "settings": {
        "defaultNodeType": "def",
        "defaultNodeColor": "#000",
        "labelSizeRatio": 0.8,
        "labelSize": "fixed",
        "defaultEdgeColor": "#fff",
        "labelThreshold": 3,
        .
        .
    }
}
```

Výpis 2: Ukázka souboru JSON

GEXF Hierarchický jazyk pro popisování komplexních síťových struktur, který vznikl v projektu Gephi. Vychází ze značkovacího jazyka XML, současně je ve verzi 1.3 a pro fungování v aplikaci je doporučená alespoň verze 1.2. Vzhledem k tomu, že vychází z jazyka XML je možné ho zpracovat v Javascriptu pomocí rozhraní DOM. [30] Pokud vrcholy neobsahují element s jejich pozicí, tak se umístí náhodně. Ukázka ve výpisu č. 3. Tento formát lze importovat i exportovat.

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.3" version="1.3" xmlns:viz="http://www.
    gexf.net/1.3/viz" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    " xsi:schemaLocation="http://www.gexf.net/1.3 http://www.gexf.net/1.3/
    gexf.xsd">
  <graph defaultedgetype="directed" mode="static">
    <nodes>
      <node id="0" label="P_EXPORT_TYPE_JAVA_METHOD">
        <viz:size value="4.0"></viz:size>
        <viz:color r="24" g="162" b="162"></viz:color>
      </node>
    </nodes>
    <edges>
```

```
      <edge id="0" source="0" target="0"></edge>
    </edges>
  </graph>
</gexf>
```

Výpis 3: Ukázka souboru GEXF;

GML Je starší přenositelný formát pro definici grafů s jednoduchou syntaxí a rozšiřitelnosti. Skládá se ze seznamu hierarchické kolekce klíč, hodnota a je podporován v mnoha aplikacích. Používá pouze ASCII znaky, proto je jednodušší na zpracování. [31] Protože jsem nenašel vhodný nástroj pro zpracování tohoto formátu, tak jsem se rozhodl pro zpracování použít knihovnu PegJs, kde jsem si definoval sadu pravidel jak zpracovat tento soubor. Ukázka ve výpisu č. 4. Tento formát lze pouze importovat.

```
graph
[
  directed 0
  node
  [
    id 0
    label "0"
  ]
  node
  [
    id 1
    label "1"
  ]
 edge
 [
  source 1
  target 0
  value 2.5
 ]
]
```

Výpis 4: Ukázka souboru GML;

4.2.4 Nastavení importu

U vlastní sítě je možnost načíst pouze její největší, maximální komponentu, kdy se prohledá rekurzivně celá síť pomocí prohledávání do hloubky, vybere komponenta s největším počtem

vrcholů a hlavní komponenta aplikace zobrazí tuto maximální komponentu, se kterou se pracuje jako s běžnou sítí.

4.2.5 Použití knihovny Sigma spolu s ReactJs

Další problém nastává při použití Sigmy spolu s Reactem. V Javascriptu jsou moduly dostupné od šesté verze ECMAScriptu. Vzhledem k tomu, že Sigma je starší knihovna, tudíž neobsahuje moduly, které by bylo možné importovat do Reactu. Ačkoliv je možné využít React i se staršími verzemi ECMAscriptu, připravili by jsme se o výhody, které vyšší verze ECMAscriptu poskytuje. React obsahuje přednastavené konfigurace, například které soubory se mají importovat, transpilovat apod. a interní moduly, které si spravuje sám. Dalším z řešení by mohlo být exportování všech konfigurací a modulů Reactu, následně jejich úprava. Operace exportu je jednosměrná, o všechnu konfiguraci se poté stará uživatel a mohou nastat problémy např. během transpilace, automatické obnovy při změně kódu apod. Proto jsem se rozhodl pro nejjednodušší a nejbezpečnější variantu. Přidal jsem Sigmu s jejími pluginy do statické složky a přidal *script* značku do statického HTML souboru. Statická složka je po transpilaci zkopírovaná k výsledným souborům. HTML soubor je zpracován vždy po otevření URL aplikace prohlížečem. Toto řešení se obecně nedoporučuje, protože React nemůže zobrazit chyby během sestavování a místo toho se uživateli zobrazí chyba, pokud soubory chybí, dále potom zmenšit jejich velikost, odstraněním nepotřebných znaků, komentářů ap. a předcházet zbytečnému opětovnému stahování souborů znovu ze sítě, pokud již byly načteny. Sigma je tedy uložena v globální proměnné *window* a vzhledem k tomu, že se jedná o jednostránkovou aplikaci, tak můžeme k ní přistupovat v každé komponentě.

4.2.6 Načítaní sítě

Po výběru sítě úvodní komponenta vytvoří objekt, který obsahuje typ vybraného souboru a BLOB URL, odkazující na objekt *File*, tedy vybranou síť. Tento objekt se předá hlavní komponentě aplikace, která podle typu souboru zvolí vhodný způsob zpracování, zkontroluje zda-li byla zvolená možnost výběru největší komponenty a pokud je soubor správný, obsahuje všechny nutné elementy a neobsahuje syntaktické chyby, tak síť načte a zobrazí. Pokud soubor nelze zpracovat, zobrazí se znovu úvodní komponenta s chybou. Síť potom můžeme dále upravovat pomocí komponenty nastavení, která mění a aplikuje nastavení na síť. Skládá se z několika kategorií, které jsou více popsány v části o uživatelském rozhraní, viz kapitola 5.

4.2.7 Skupiny

Skupiny ulehčují manipulaci, kdy se nemusí nastavovat vrcholy jednotlivě, ale lze aplikovat nastavení na skupinu. Skupiny je možné tvořit jak vlastní, výběrem nástroje pro výběr vrcholů z nastavení, kdy se nad sítí vytvoří nové plátno, na které lze nakreslit mnohoúhelník. Po nakreslení se zkontrolují souřadnice všech vrcholů v síti a pokud je jejich umístnění v tomto mnohoúhelníku,

tak je vytvořena nová skupina s těmito vrcholy. Jeden vrchol může patřit do více skupin. Dále pomocí detekce komunit. Po detekci se vytvoří skupiny pro každou komunitu a obarví vrcholy jednotlivých komunit. Na výběr barev je použita již zmíněná knihovna iwanthue. Každá skupina má číselné označení podle jejího pořadí, toto označení lze změnit a barvu rovněž, barva je použita podle barvy prvního vrcholu v dané skupině. Je možno zobrazit i jejich zabranou plochu v sítí. U zabrané plochy v sítí se snažíme najít konvexní obal vrcholů v dané skupině a jako barva pozadí je použitá barva skupiny.

4.2.8 Louvain detekce komunit

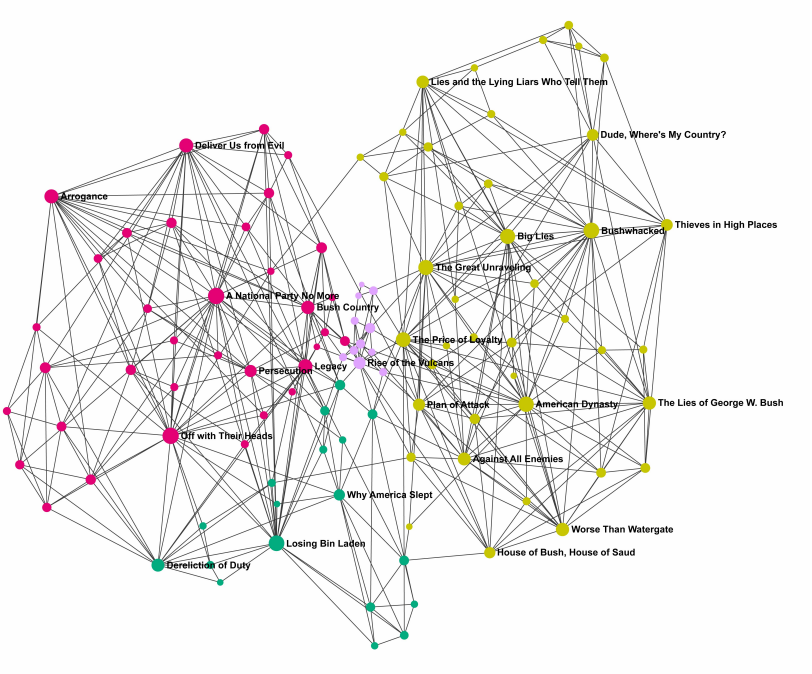
Louvain metoda pro detekci komunit je jednoduchá, efektivní a snadno implementovatelná metoda pro identifikaci komunit ve velkých sítích. Zjednodušeně, algoritmus je rozdělen na dvě fáze. Nejprve vytvoříme pro každý vrchol komunitu. Každý vrchol je ve své vlastní komunitě. Poté se pro každého souseda vrcholu vyhodnotí, jakou by získal modularitu, kdyby se přesunul ze své komunity do komunity souseda. Vrchol je přesunut do komunity s největší kladnou modularitou, pokud nezíská žádnou kladnou modularitou tak zůstává ve své komunitě. Tento proces se opakuje pro všechny vrcholy do doby, dokud již není možné získat lepší výsledek, neexistuje lepší modularita pro vrcholy. Tímto je ukončena první fáze. Ve druhé fázi se tvoří síť, kde vrcholy nové sítě jsou komunity vytvořené v první fázi. Váhy hran mezi novými vrcholy jsou dány jako součet vah hran vrcholů ve dvou odpovídajících komunitách. Počet komunit se snižuje a výpočetní čas je menší s každým opakováním těchto dvou fází. Opakování fází probíhá do doby, dokud již není možné získat lepší modularitu a nenastávají změny v komunitách. [27]

4.2.9 Implementované metody vizualizace

Aplikovat je možné následující metody vizualizace. K vybraným metodám vizualizace jsou zde i graficky exportované ukázky z této aplikace.

- Force-based
  - ForceAtlas2

    Tento force-based algoritmus (viz kapitola 2.2.1) vznikl v projektu Gephi (viz kapitola 3.1.1) z potřeb uživatelů, kde jeden z požadavků byl, aby algoritmus běžel nepřetržitě a k jeho zastavení dojde až s požadavkem uživatele, kvůli tomu, že se sítí lze pracovat a změny se projeví okamžitě. Algoritmus simuluje fyzikální systém, kde se vrcholy navzájem odráží jako nabité částice, zatím co hrany přitahují jejich vrcholy jako pružiny. Tyto síly vytvářejí pohyb, který konverguje do vyváženého stavu. Proces výpočtu pozice vrcholu je závislý pouze na spojeních hran a vrcholů, atributy vrcholu nejsou brány v potaz. Výsledek závisí na počátečním stavu sítě, může tedy skončit ve skorém stádiu. Cílem tohoto algoritmu je přiblížit vrcholy menšího stupně k vrcholům s vyšším stupněm, tzn. vyladit odpudivou sílu mezi vrcholem s vyšším stupněm a



Obrázek 11: Síť hypertextových odkazů mezi web blogy o americké politice, zaznamenaná v roce 2005 s použitím ForceAtlas2 algoritmu

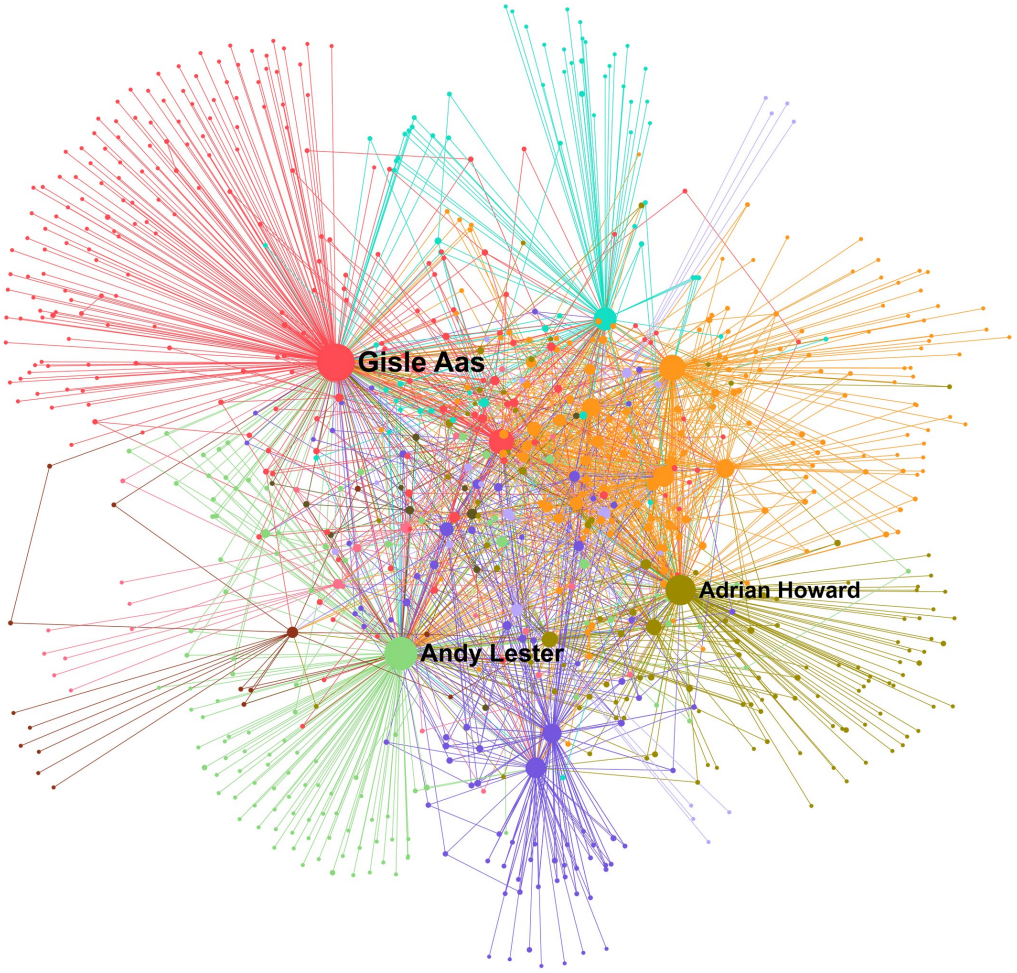
vrcholem menšího stupně, takže ve vyváženém stavu budou blíž k sobě. Přitažlivá síla je vypočítávána ze vzdálenosti jednotlivých vrcholů a pokud je síť vážená, tak je síla úměrná váze hrany. [32]

ForceAtlas2 (viz obr. 11), který již existuje jako plugin pro Sigmu včetně BurnesHut optimalizace, funguje paralelně, na dvou vláknech. Hlavní vlákno vytvoří nové vlákno (pracovníka) (kapitola 4.1.2), které dostane pouze objekty nutné na výpočet. Poté, co se výpočet provede, se vypočtené pozice vrátí hlavnímu vláknu, protože pouze hlavní vlákno může změnit pozice vrcholů podle vypočtených hodnot.

– Fruchterman-Reingold

Force-based algoritmus (viz kapitola 2.2.1), který je inspirovaný molekulární a planetární simulací. V tomto algoritmu jsou vrcholy reprezentovány jako atomické částice a vyvíjejí na sebe přitažlivé a odpudivé síly, které vyvolávají pohyb. Velikost pohybu je omezená maximální hodnotou a tato hodnota se zmenšuje s časem tak, jak je rozložení lepší a pohyb vrcholů jemnější. Odpudivá síla je vypočítávaná mezi každým párem vrcholů, ale přitažlivá síla se počítá pouze se sousedními vrcholy. Tímto můžeme snížit čas výpočtu. Pro vizualizaci jsou nutné dva principy. První je, aby vrcholy spojené pomocí hrany byly blíž k sobě a druhá, aby vrcholy nebyly příliš blízko sebe. Blízkost vrcholů je určená podle volného místa a jejich počtu. [33]

Obdobně jako předchozí metoda funguje i Fruchterman-Reingold (viz obr. 12), kde jsem se inspiroval předchozí implementací, takže během spuštěného algoritmů je vidět



Obrázek 12: Síť zobrazující vztahy mezi vývojáři a balíčky v jazyku Perl s použitím Fruchterman-Reingold algoritmu

změna s každou iterací. Oproti předchozí metodě má tato metoda stanovený počet opakování.

- Náhodné

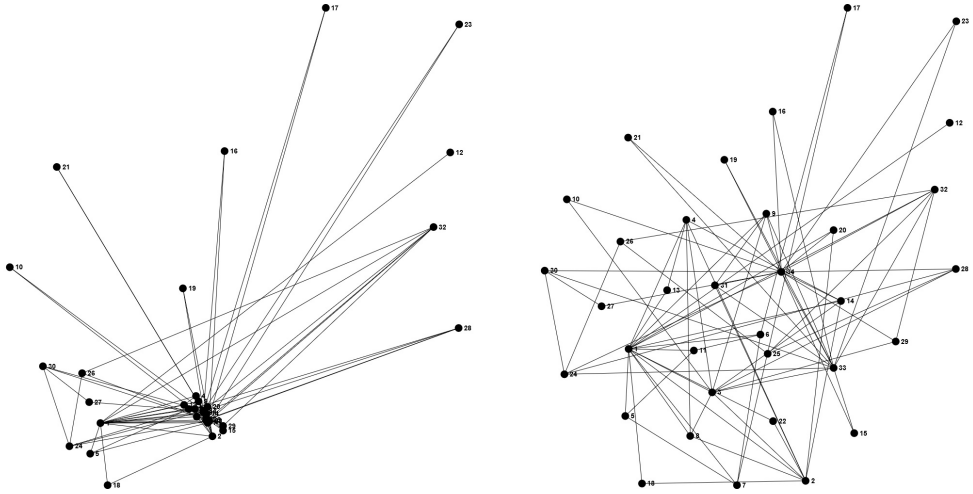
  Vrcholy grafu jsou umístěny náhodně, v rozmezí od 0 až 1, pro jednotlivé souřadnice.

- Kruhové zobrazení

  Vrcholy jsou za sebou, podle skupin, umístěny rovnoměrně na kružnici. Viz obrázek 6 na straně 17.

- Bez překrývaní vrcholů

  Distribuce vrcholů sítě tak, aby se navzájem nepřekrývali, případně aby kolem jednotlivých vrcholů byla definovaná mezera. Viz obrázek 13, který zobrazuje síť před a po použité tohoto zobrazení s mezerou mezi vrcholy alespoň 50 bodů.



(a) Před použitím zobrazení bez překrývaní vrcholů s mezerou mezi vrcholy

(b) Po použití zobrazení bez překrývaní vrcholů s mezerou mezi vrcholy

Obrázek 13: Zachary klub síť

- Rotace

  Natočení celé sítě o daný úhel.

4.3 Diagram případů užití

Diagram případů užití (obrázek číslo 14) obsahuje i popis jednoho scénáře (tabulka 1), který popisuje aplikaci vybrané metody zobrazení na síť. Vzhledem k jednoduchosti aplikace jsem uvedl pouze tento jeden scénář, protože obdobně by vypadaly i ty další.

4.4 Diagram komponent

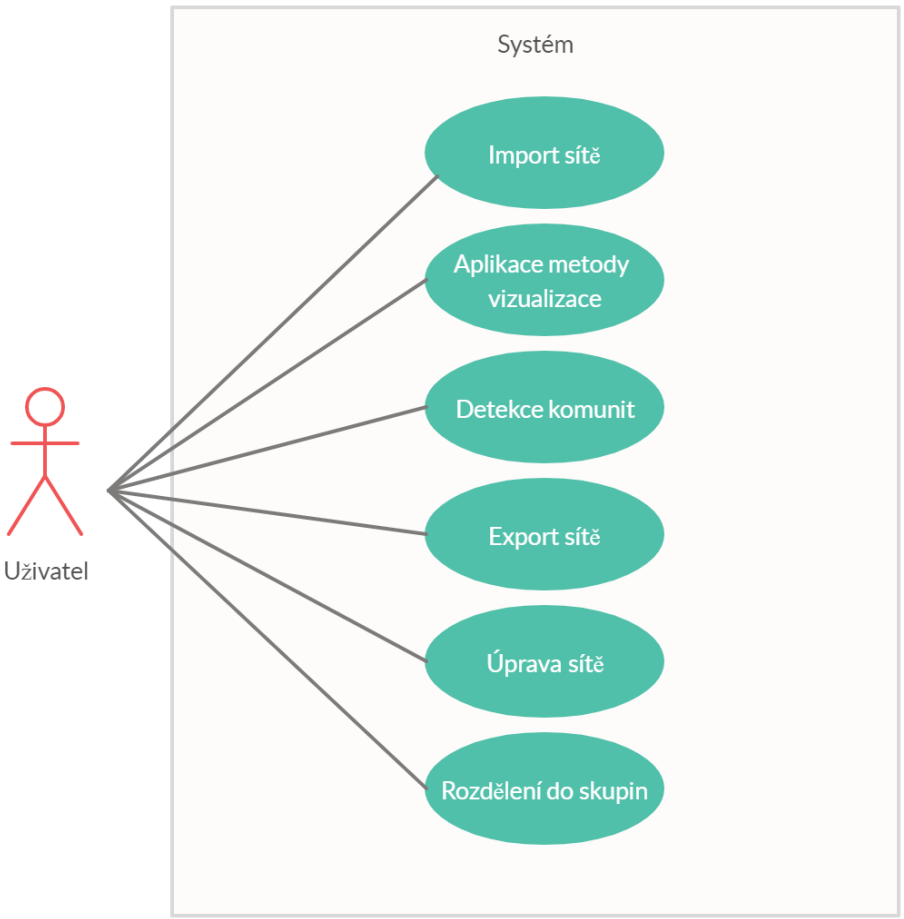
Rozdělení systému na komponenty, kde světle modrá barva reprezentuje komponenty třetích stran, tj. externí knihovny a světle žlutá barva reprezentuje moje komponenty, tedy jednotlivé React komponenty, viz obrázek číslo 15.

4.5 Nasazení

Pro spuštění aplikace, která již běží na nějakém serveru potřebujeme pouze internetové připojení a webový prohlížeč Chrome ve verzi 29 a výše, kvůli podpoře File API a použitým CSS stylům. Pokud by jsme chtěli aplikaci spustit lokálně nebo upravit zdrojový kód, tak musíme splnit systémové požadavky. Podrobnější instrukce jsou obsaženy v příloze A.

4.5.1 Systémové požadavky

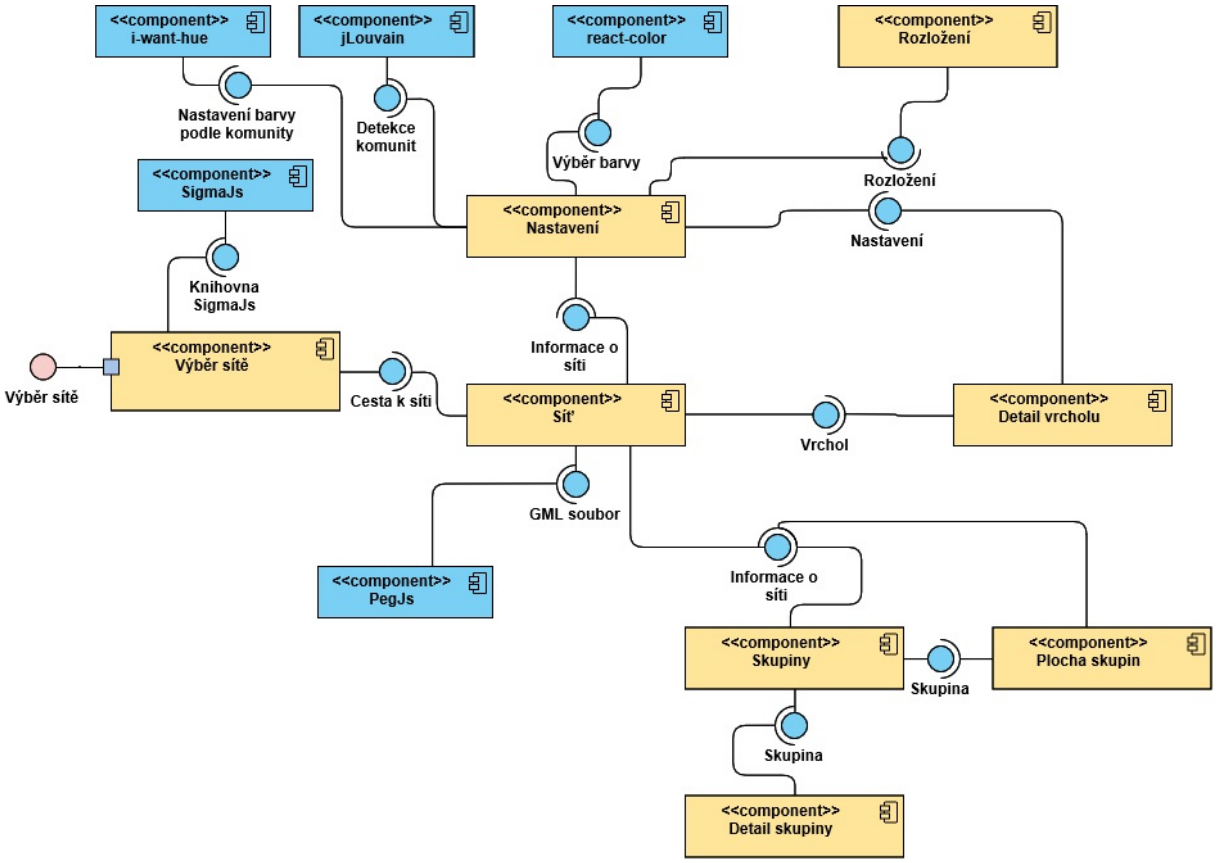
- 32bit nebo 64bit CPU s taktem alespoň 1 GHz



Obrázek 14: Diagram případů užití

Tabulka 1: Scénář aplikace vybrané metody

Popis	Aplikace vybrané metody vizualizace
Aktéři	Uživatel Systém
Podmínky před spuštěním	Uživatel musí mít naimportovanou síť.
Průběh	1. Uživatel si vybere požadované zobrazení z možností v nastavení 2. Systém zobrazí dostupné nastavení zobrazení a možnost pro spuštění 3. Uživatel nastaví požadované nastavení 4. Uživatel spustí aplikaci zobrazení 5. Systém provede změnu zobrazení
Alternativní průběh	Alternativní průběh 1 5.1 Uživatel zastaví průběh zobrazení Alternativní průběh 2 5.1 Uživatel změní požadované zobrazení 5.2 Systém zastaví aktuálně spuštěné zobrazení
Podmínky po ukončení	Síť je změněná podle vybrané vizualizace.



Obrázek 15: Diagram komponent

- 4 GB a více RAM
- Operační systém novější než následující
  - GNU/Linux: kernel 2.6.32
  - macOS: 10.10
  - Windows: Windows 7 / 2008 R2
- Webový prohlížeč Chrome, minimálně ve verzi 29
- NodeJs 12.16.0 LTS a novější

4.5.2 NodeJs

Umožňuje spouštět Javascriptový kód mimo webový prohlížeč a pracovat se systémovými prostředky pro tvorbu škálovatelných webových aplikací. Využívá stejné jádro pro interpretaci Javascriptu jako prohlížeč Chrome. Obsahuje smyčku událostí, která provádí neblokující IO operace na úrovni systému, rozděluje úlohy pomocí vláken, takže dosahuje lepšího výkonu a bezpečnosti. [34]

5 Popis uživatelského rozhraní

Hlavní nastavení, které se nachází na pravé straně, je rozděleno na několik kategorií. Nastavení je aplikováno ihned po změně hodnoty, kromě nastavení spuštěné metody zobrazení. Na levé straně se nachází seznam skupin. Oba panely je možno skrýt po kliknutí na šipku vedle jednotlivých panelů. Na obrázku s číslem 16 je ukázka výsledné aplikace s načtenou sítí, detekovanými komunitami a aplikovaným rozložením ForceAtlas2 (viz kapitola 4.2.9).

Domů Slouží pro návrat na úvodní obrazovku, pro výběr nové sítě.

Výběr vrcholů Vytvoří nad sítí novou vrstvu, na které lze nakreslením vybrat požadované vrcholy. Po nakreslení se zkontroluje pozice nakresleného mnohoúhelníku a jednotlivých vrcholů, pokud jsou pozice vrcholů na ploše mnohoúhelníku, tak jsou přidány do nové skupiny. Takto lze vytvářet vlastní skupiny. Během výběru nelze přibližovat ani posouvat síť.

Export Slouží pro export sítě do zvoleného formátu, jak už formát, se kterým můžeme dále pracovat v jiných aplikacích, tak i JSON, včetně některých nastavení, které se při opětovném importu do aplikace použijí. Grafický výstup, buď do rastrového formátu PNG nebo vektorového formátu SVG. U grafických formátů je možno skrýt popisky, nastavení je aplikováno podle toho, jestli je vybraná možnost zobrazit popisky v kategorii nastavení Popisek. Název exportovaného souboru je stejný jako název importovaného souboru sítě.

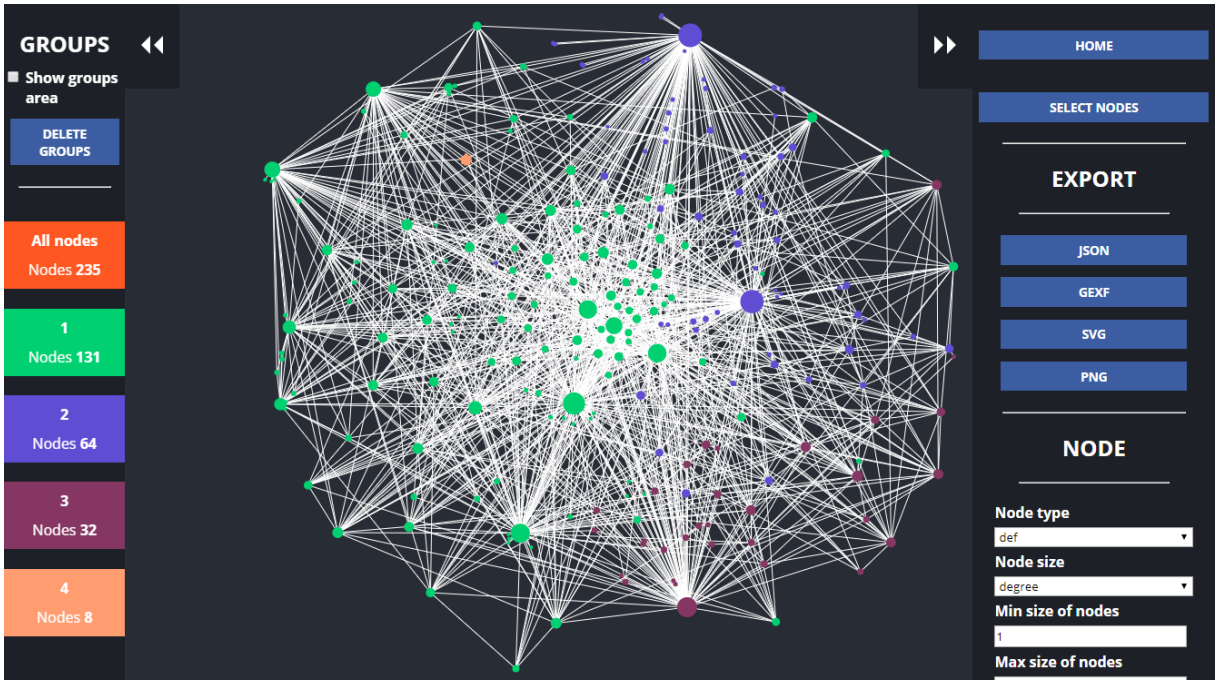
Vrchol Nastavení pro změnu vlastností všech vrcholů v sítí, barva, grafický typ vrcholů a velikost vrcholů absolutně, nebo relativně k ostatním vrcholům. Po kliknutí na konkrétní vrchol je možné ho odstranit nebo měnit pouze jeho nastavení, avšak jeho nastavení jsou omezenější.

Popisek Nastavení pro změnu všech popisků, jejich skrytí, barva, pevná, nebo proporcionální velikost závislá na velikosti vrcholu a měřítko, kdy se má zobrazit popisek, vypočtené podle velikosti vrcholu a aktuálního přiblížení sítě.

Hrana Nastavení všech hran, barva, tvar, velikost, pevná, nebo podle velikosti zdrojového vrcholu, možnost zobrazit směr hrany a nastavení relativní velikosti v závislosti k ostatním hranám.

Komunity Umožňuje zvolit si algoritmus pro detekci komunit. Aktuálně obsahuje pouze možnost pro Louvain algoritmus. Pro použití stačí mít naimportovanou síť, nevyžaduje žádné specifické nastavení a vytvoří jednotlivé skupiny, podle počtu detekovaných komunit.

Rozložení Každé rozložení má své nastavení, které je zobrazeno a použito při jeho spuštění, tzn. pokud změníme nastavení a rozložení již běží, tak musíme zastavit spuštěné rozložení a spustit ho znovu. Toto omezení je kvůli tomu, že novému vláknu se musí předat nové



Obrázek 16: Ukázka celé aplikace s načtenou sítí

nastavení. Pokud spuštěné rozložení běží a chtěli bychom změnit rozložení na jiné, tak se aktuálně spuštěné rozložení zastaví.

Skupiny Obsahuje informaci o jednotlivých skupinách, umožňuje odstranit všechny skupiny, změnit konkrétní nastavení a zobrazit plochu, kterou zabírá konkrétní skupina. Skupiny je možno tvořit jak ručně, výběrem vrcholů, tak automaticky, pomocí detekce komunit. Každou nově vytvořenou skupinu lze pojmenovat a u vrcholů je možno aplikovat určitá nastavení, alternativa pro vybíraní vrcholů po jednom. Konkrétně je možno skupinám nastavit název a vrcholům ve skupině vlastnosti jako barvu, velikost, tvar, odstranit skupinu a exportovat skupinu jako samostatný graf ve formátu JSON, se kterým lze pracovat jako se samostatnou sítí.

6 Závěr

Podařilo se mi vytvořit webovou aplikaci, která si myslím, splňuje požadované náležitosti, tedy obsahuje možnost načíst síť a na tu aplikovat metodu vizualizace, případně pomocí uživatelského rozhraní měnit nastavení vybraného rozložení, vrcholů nebo hran. Během vytváření této aplikace jsem narazil na spoustu pro mne nových poznatků o sítích, jejich reprezentaci, jednotlivých možnostech zobrazení nebo detekování shluků v sítích, o kterých jsem moc nevěděl. Během hledání existujicích řešení jsem si vyzkoušel práci s aplikacemi, které se používají pro práci se sítěmi i některé pro mne nové knihovny. Z technického hlediska se mi s knihovnou Sigma a jejímy pluginy pracovalo příjemně, hlavně kvůli jednoduchosti, ale asi bych u podobného projektu nepoužil znovu React, ale knihovnu odvozenou z Reactu, která podporuje možnost úpravy některých konfiguračních souborů, především kvůli podpoře web workerů nebo jiných typu souborů. Možnou nevýhodou je, že samotný výpočet jednotlivých zobrazení neprobíhá paralelně na vícero vláknech, aby se urychlila doba běhu zobrazení, ustálení sítě. Aplikace by šla dále rozšířit o další funkce pro analýzu sítě, které například poskytuje Gephi, tato aplikace v podstatě žádné informace o síti neposkytuje, aby mohla sloužit jako alternativa pro již existující aplikace. Potom by stála za zvážení knihovna CytoscapeJs, pro komplexnější analýzu. Rovněž by grafický styl celé aplikace, který je v aplikaci od její první verze skoro stejný, mohl být přívětivější.

Literatura

1. BARABÁSI, Albert-László et al. *Network science*. Cambridge university press, 2016.
2. EASLEY, David; KLEINBERG, Jon. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press, 2010. Dostupné z DOI: 10.1017/CBO9780511761942.
3. FORTUNATO, Santo. Community detection in graphs. *Physics reports*. 2010, roč. 486, č. 3-5, s. 75–174.
4. Brno: Institut biostatistiky a analýz LF a PřF MU, 2014. Dostupné také z: https://portal.matematickabiologie.cz/index.php?pg=zaklady-informatiky-pro-biology--teoreticke-zaklady-informatiky--teorie-grafu--reprezentace-grafu.
5. *Graph Representation: Adjacency List and Matrix*. Dostupné také z: https://algorithmtutor.com/Data-Structures/Graph/Graph-Representation-Adjacency-List-and-Matrix/.
6. TAMASSIA, Roberto. *Handbook of graph drawing and visualization*. Chapman a Hall/CRC, 2013.
7. KOBOUROV, Stephen G. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011*. 2012.
8. KOBOUROV, Stephen G. *Force-Directed Drawing Algorithms*. 2004.
9. *Gephi Datasets*. Dostupné také z: https://github.com/gephi/gephi/wiki/Datasets.
10. KIEFFER, Steve; DWYER, Tim; MARRIOTT, Kim; WYBROW, Michael. Hola: Human-like orthogonal network layout. *IEEE transactions on visualization and computer graphics*. 2015, roč. 22, č. 1, s. 349–358.
11. FREIVALDS, Kārlis; GLAGOĻEVS, Jans. Graph compact orthogonal layout algorithm. In: *International Symposium on Combinatorial Optimization*. 2014, s. 255–266.
12. COMMONS, Wikimedia. *File:Morse code tree3.png — Wikimedia Commons, the free media repository*. 2019. Dostupné také z: https://commons.wikimedia.org/w/index.php?title=File:Morse_code_tree3.png&oldid=379691186. [Online; accessed 22-March-2020].
13. Layered Graph Drawing (Sugiyama Method), s. 54. Dostupné také z: http://www.it.usyd.edu.au/~shhong/fab.pdf.
14. *YWorks*. Dostupné také z: https://www.yworks.com/pages/layered-graph-layout.
15. HOLTZ, Yan. *Data to viz: ARC DIAGRAM*. Dostupné také z: https://www.data-to-viz.com/graph/arc.html.
16. *Gephi*. 2017. Dostupné také z: https://gephi.org/.
17. *Pajek*. Dostupné také z: http://mrvar.fdv.uni-lj.si/pajek/.

18. *Cytoscape*. Dostupné také z: https://cytoscape.org/.

19. *RAW Graphs*. Dostupné také z: https://rawgraphs.io/.

20. *Rhumbl*. Dostupné také z: https://rhumbl.com/.

21. *D3.js*. Dostupné také z: https://d3js.org/.

22. *Cytoscape Web*. Cytoscape, 2007. Dostupné také z: http://cytoscapeweb.cytoscape.org/.

23. *Cytoscape JS*. Cytoscape, 2020. Dostupné také z: https://js.cytoscape.org/.

24. *Web Worker API*. Dostupné také z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API.

25. *ReactJs*. Dostupné také z: https://reactjs.org/.

26. *Iwanthue*. Dostupné také z: https://medialab.github.io/iwanthue/theory/.

27. BLONDEL, Vincent D; GUILLAUME, Jean-Loup; LAMBIOTTE, Renaud; LEFEBVRE, Etienne. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*. 2008, roč. 2008, č. 10, s. P10008.

28. *PEG.js*. 2017. Dostupné také z: https://pegjs.org/.

29. *W3C File API*. W3C, 2020. Dostupné také z: https://www.w3.org/TR/FileAPI/.

30. *GEXF*. Dostupné také z: https://gephi.org/gexf/format/index.html.

31. HIMSOLT, Michael. GML: A portable Graph File Format. In: 2010.

32. JACOMY, M. Venturini, T.-Heymann, S.-Bastian, M.[2014]: ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PLoS ONE*. Roč. 9, č. 6.

33. FRUCHTERMAN, Thomas MJ; REINGOLD, Edward M. Graph drawing by force-directed placement. *Software: Practice and experience*. 1991, roč. 21, č. 11, s. 1129–1164.

34. *NodeJs Docs*. OpenJS Foundation, 2020. Dostupné také z: https://nodejs.org/en/docs/.

A Popis nasazení

Pokud splňujeme systémové požadavky, máme stažený a rozbalený projekt, tak se můžeme v příkazové řádce dostat do kořenové složky projektu, kde se nachází soubor *package.json*, který kromě informací o projektu, názvu, autorovi nebo licenci obsahuje i informace o balíčcích, jak už vývojových, které jsou nutné pro vývoj a testováni, tak i o těch, které jsou nutné pro běh aplikace a to včetně jejich verze, která by měla zaručit, že aplikaci spustíme i v době, kdy došlo k aktualizaci balíčků.
Nyní můžeme spustit příkaz

npm install

kterým nainstalujeme všechny potřebné balíčky a jejich závislosti definované v *package.json*. Po dokončení instalace můžeme přejít ke spuštění, pomocí příkazu

npm start

který spustí vývojovou verzi, webový server, provádí automatickou transpilaci a znovu načítaní v případe změny kódu nebo zobrazuje chyby a upozornění během sestavování. Dostupná z webového prohlížeče, IP adresa a port bude uveden v příkazové řádce, nejčastěji ale na adrese

127.0.0.1:3000

a lokální adrese prvního rozhraní, pokud je počítač připojený do nějaké sítě a není uvedeno jinak v konfiguračním souboru.

Pokud by jsme chtěli aplikaci nasadit na vlastním serveru, je nutné použít přikaz

npm build

který vytvoří tzv. minifikovanou verzi, kdy se ze zdrojových souborů jak naší aplikace, tak i nutných knihoven pro běh odstraní všechny nepotřebné znaky, bílé znaky nebo komentáře, aby se zmenšila jejich velikost. Hlavní HTML soubor pro nasazenení na vlastním serveru se všemi potřebnými soubory bude ve složce *build*. V základu, pokud spustíme *index.html* ve webovém prohlížeči, bez nějakého serveru, uvidíme bílou obrazovku, protože všechny potřebné soubory mají nedefinovanou relativní cestu. Proto, pokud chceme aplikaci i spustit, tak nejjednodušší možností je nainstalovat globálně balíček *serve* příkazem

npm install -g serve

poté se dostat v příkazové řádce opět do kořenového adresáře projektu a do složky build, ve které spustíme příkaz

serve -s build

zobrazí se nám IP adresy a port na kterém by měla být již funkční aplikace dostupná.